TARGET CLASSIFICATION VIA SUPPORT VECTOR MACHINES

Robert E. Karlsen, David J. Gorsich and Grant R. Gerhart

U.S. Army Tank Automotive Research, Development and Engineering Center
Warren, MI 48397-5000

ABSTRACT

The area of automatic target classification has been a difficult problem for many years. Many approaches involve extracting information from the imagery through a variety of statistical filtering and sampling techniques, resulting in a reduced dimension feature vector, which can then be input to a learning algorithm. In this paper, we outline a method that is virtually independent of feature vector size and can therefore be applied to entire images, so that the feature extraction and learning algorithm are combined into one. We present the results of two image classification tests, both of which yielded excellent results.

Introduction

The problem that we address in this paper is image classification. Specifically, we consider algorithms that can take as input a digital image and classify it according to some criterion. Often this is a three-step process, consisting of pre-processing, feature extraction, and decision algorithm. The pre-processing is used to remove redundant information or to transform the image to a space where the objects are more easily classified. We have used the multiresolution approach in previous applications, implemented through the fast wavelet transform[1-3]. The feature extraction step is employed to reduce the dimensionality of the problem. Examples of features include peaks in the Fourier spectrum, statistical measures of edge densities, multiresolution energies or a histogram of gray levels. The final step is the decision module, which takes as input the lower dimension feature vector and outputs the classification. Often the decision algorithm is also a learning algorithm. In this case, a sufficiently large number of sample images, with their associated classification, are presented to the algorithm. The algorithm then adjusts certain parameters in order to satisfy a minimum error criterion.

The typical learning algorithm is a neural network (fuzzy logic systems generally have an embedded neural network for automated learning). The reason that feature vector selection is so important for neural networks (specifically back-propagation neural networks) is that the complexity of the network scales with the size of feature vectors. In fact, the number of free parameters that must be determined is proportional to the size of the feature vector and is often many times larger. This generally necessitates a large number of training samples in order to constrain the error minimization sufficiently.

The Support Vector Machine (SVM) algorithm[4-6] avoids many of these problems. Here, only the dot product between feature vectors enters the problem. Therefore, the length of the feature vector has little effect on the computational complexity of the algorithm. By design, SVM is a large margin classifier, and can give reasonable results even for sparse training sets, where the number of samples may be less than the size of the feature vector. SVM can also be made resistant to outliers of a given size, by adjusting a cost parameter.

The paper begins by providing a brief tutorial on Support Vector Machines and outlining derivations of some of the important elements. We then use the SVM algorithm on two data sets, the first is a standard handwritten digit data set[7], derived from two NIST data sets, and the second is a collection of images of military vehicles. Both of these were classified accurately with SVM. We conclude by summarizing our results and indicating the future direction of our research.

Support Vector Machines

To introduce the Support Vector Machine (SVM), we first consider a linearly separable problem; i.e. the data can be separated completely by a hyperplane. Figure 1 shows an example in two dimensions, where the hyperplane is a line. The object is to find the best hyperplane that separates the data into two classes, where, by 'best' we mean the hyperplane that gives the best classification results when new data is used.

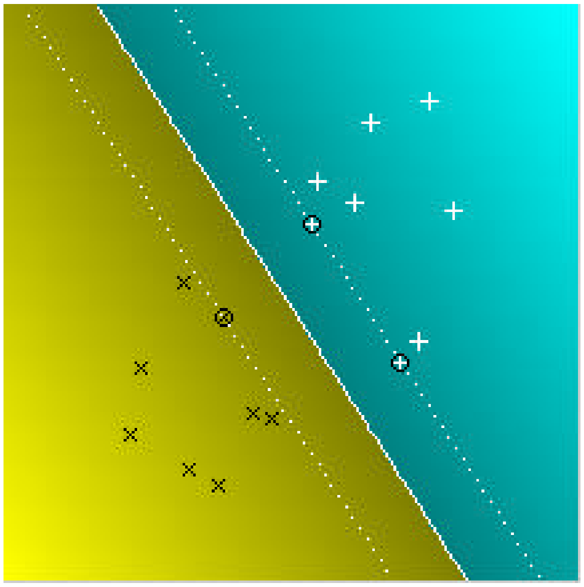

Fig. 1: Linearly separable data.

The SVM algorithm[4-6] is based on finding a pair of parallel hyperplanes, which separate the data and which have the largest perpendicular distance between them. It is conjectured that this will provide a good approximation to the 'best' separating hyperplane. Each data point is described by a feature vector $\vec{x}$ and a truth value *y*, the latter of which can take the values of +1 or –1, depending on the class. The two hyperplanes are required to pass through at least one point of each class and there can be no points between them. The boundary between the classes is then defined to be a third parallel hyperplane that is halfway between the other two. The data points that the outer hyperplanes pass through, which are circled in Fig. 1, are called the support vectors, the meaning of which will be explained later. The two outer hyperplanes are described by the following expressions,

$$\begin{aligned}\vec{w}\cdot\vec{x}+b&=+1,\\ \vec{w}\cdot\vec{x}+b&=-1,\end{aligned}\qquad(1)$$

with the first going through a point of class *y*=+1 and the second going through a point of class *y*=–1. The constants $\vec{w}$ and *b* define the hyperplanes, with $\vec{w}$ being normal to the hyperplanes and $-b/\,||\vec{w}\,||$ being the perpendicular distance from the origin to the middle hyperplane. The RHS of Eq. (1) will be greater than or equal to +1 for all points of class *y*=+1 and will be less than or equal to –1 for all points of class *y*=–1. These can be combined into the following constraint on all the data points,

$$y_i\left(\vec{w}\cdot\vec{x}_i+b\right)-1\geq 0\,.\qquad(2)$$

The perpendicular distance between the two outer hyperplanes is equal to $2\,/\,||\,\vec{w}\,||$. Therefore, finding the hyperplanes with the largest margin reduces to finding values for $\vec{w}$ and *b* that minimize $||\,\vec{w}\,||^2$, subject to the constraint in Eq. (2).

A standard method for handling optimization problems with constraints is through the minimization of a Lagrangian[8-10]. The constraints are taken into account by adding terms involving Lagrange multipliers to the objective function. In this case, this results in the following primal Lagrangian[4-6],

$$L_P=\frac{1}{2}\,||\vec{w}\,||^2-\sum\quad{}_i\,y_i\left(\vec{w}\cdot\vec{x}_i+b\right)+\sum\quad{}_i\qquad(3)$$

where $\;{}_i$ are the Lagrange multipliers associated with each of the constraints in (2). Notice that at the boundary of the constraint equation (2), the extra terms added in (3) are zero. As one moves away from the boundary, the Lagrangian becomes smaller. Because the constraints are inequalities, bounded from below, the Lagrange multipliers are required to be non-negative.

Setting the derivative of the Lagrangian in (3) with respect to $\vec{w}$ and b (the primal variables) equal to zero, results in the following expressions,

$$\vec{w} = \sum \;_i y_i \vec{x}_i \,, \tag{4a}$$
$$\sum \;_i y_i = 0 \,, \tag{4b}$$

while from the definition of the Lagrange multipliers, we obtain,

$$_i \left(y_i \left(\vec{w} \cdot \vec{x}_i + b \right) - 1 \right) = 0 \,. \tag{4c}$$

Inserting Eqs. (4a) and (4b) into (3), results in the dual Lagrangian,

$$L_D = \sum \;_i - \frac{1}{2} \sum \;_i \;_j y_i y_j \vec{x}_i \cdot \vec{x}_j \,. \tag{5}$$

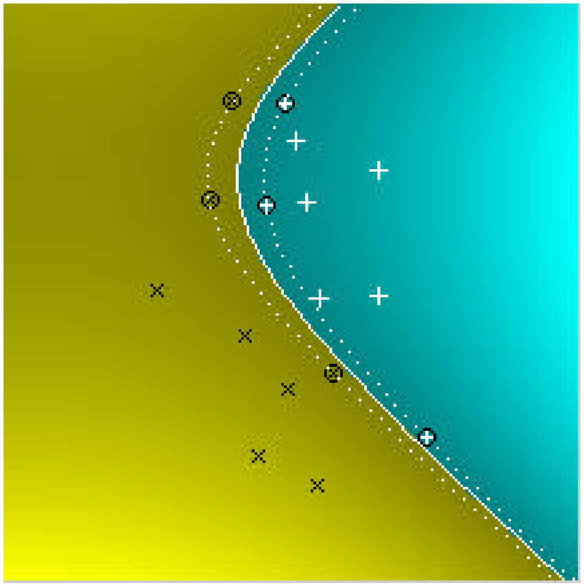

Fig. 2: Non-linearly separable data.

The problem is now reduced to finding the Lagrange multipliers (the dual variables) that maximize Eq. (5) and satisfy both the non-negativity constraints and the constraints of Eq. (4b). Equation (4c) means that only those data points which lie on the outer hyperplanes (and hence are active constraints) will have non-zero Lagrange multipliers. These data points are called the support vectors and they are the points that determine the position of the hyperplanes. One can move the other points around the feature space or remove them entirely and the solution will not change, provided one does not move a point across one of the outer hyperplanes.

The relationships in Eqs. (4), together with the constraints in Eq. (2) and the non-negativity constraints on the Lagrange multipliers make up what are known as the Karush-Kuhn-Tucker[8-12] (KKT) conditions for this particular problem. The KKT conditions are general rules that arise in constrained optimization problems, and finding solutions that obey them generally results in an optimal solution. Since SVM is a quadratic-programming problem (the objective function, $\| \vec{w} \|^2$, is quadratic) with linear constraints, the KKT conditions are necessary and sufficient for the resulting $\vec{w}$, b and α to be an optimal solution. One can solve Eq. (5) using any quadratic programming solver, although different solvers perform better on different types of problems[6,8-10]. Solving the quadratic programming problem is actually one of the most difficult parts of SVM and will not be discussed further in this paper.

Once the Lagrange multipliers are known, the solution for $\vec{w}$ is given by Eq. (4a), where the sum is over the support vectors, since they are the only ones with non-zero . One can find b from Eq. (4c), using any of the support vectors, although one generally averages over all the support vectors for better accuracy. Once these constants are known, the classification of an unknown vector, $\vec{v}$, is given by the sign of,

$$b + \sum \;_i y_i \vec{x}_i \cdot \vec{v} \,, \tag{6}$$

where the sum is over the support vectors. This determines on which side of the boundary (or middle) hyperplane that the data point falls.

Now suppose that the boundary between the data is nonlinear. An example of this situation is shown in Fig. 2. One cannot separate the two classes with a straight line. The structure of the SVM equations allows a simple solution to this situation. Map the data, through a nonlinear transformation , to a different space, where the data can be separated with a hyperplane. This results in the Lagrangian in (5) being transformed to,

$$L_D = \sum \;\; _i - \frac{1}{2}\sum \;\; _i \;\; _j y_i y_j \;\; (\vec{x}_i)\cdot \;\; (\vec{x}_j)\,, \tag{7}$$

and the classification relation in Eq. (6) becomes,

$$b + \sum \;\; _i y_i \;\; (\vec{x}_i)\cdot \;\; (\vec{v})\,. \tag{8}$$

Since Eqs. (7) and (8) depend only on the dot product between the two transformed feature vectors, one can employ a kernel function,

$$K(\vec{x},\vec{y}) = \;\; (\vec{x})\cdot \;\; (\vec{y})\,, \tag{9}$$

and never need to computer the transformation explicitly. Equation (8) then becomes,

$$b + \sum \;\; _i y_i K(\vec{x}_i,\vec{v})\,, \tag{10}$$

with the test feature vector now inside the summation over the support vectors.

In general, the mapping will be to a higher dimensional space. For example, suppose the data is in two dimensions and the kernel is $K(\vec{x},\vec{y}) = (\vec{x}\cdot\vec{y})^2$, then the mapping could be to three dimensions (but not to two) with either of the transformations (among others)[5],

$$(\vec{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix} \quad \text{or} \quad (\vec{x}) = \frac{1}{\sqrt{2}}\begin{pmatrix} x_1^2 - x_2^2 \\ 2x_1x_2 \\ x_1^2 + x_2^2 \end{pmatrix}, \tag{11}$$

or the mapping could be to four dimensions,

$$(\vec{x}) = \begin{pmatrix} x_1^2 \\ x_1x_2 \\ x_1x_2 \\ x_2^2 \end{pmatrix}, \tag{12}$$

where the subscripts in all cases refer to components of the vector $\vec{x}$. The kernel remains the same in each case and the resulting classification results are identical. Since one is still solving the linear problem, just in a different space, the computational overhead is essentially the same. The solution and parameters for the hyperplane are in the higher dimensional space and when one transforms back to the original space the boundary becomes nonlinear. However, there is, in general, no way to analytically invert the solutions for $\vec{w}$ and *b*. Hence, one must use Eq. (10) to classify test feature vectors.

The advantage to using the kernel approach is that the higher dimensional (or embedding) space is essentially hidden from the user. One, in fact, never needs to know the function . It could even be of infinite dimension. The disadvantage is that one must use Eq. (10) to classify each test vector and if there are a large number of support vectors, the testing phase can be somewhat slow. On the other hand, for low dimension problems, one could work in the embedding space and compute $\vec{w}$ once, via Eq. (4a), with the support vectors also transformed to

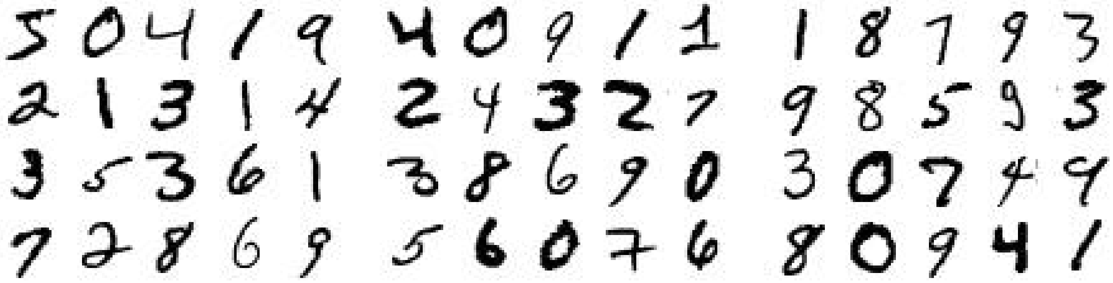

Fig. 5: Digits 1-60 from the MNIST training database.

the embedding space, as in Eq. (8). Then one can simply transform the test vector to the embedding space and compute a single dot product to classify a feature vector. The problem is dimensionality. For example, with the homogenous polynomial kernel, $K(\vec{x},\vec{y}) = (\vec{x}\cdot\vec{y})^{p}$, the dimension of the embedding space is[§],

$$\frac{(d+p-1)!}{(d-1)!\,p!}, \qquad (13)$$

where d is the dimension of the feature vector. With a 10x10 image, the feature vector has 100 elements and, using a kernel with p=2, results in the dimension being 5,050. However, with p=4, the dimension becomes 4,421,275 and so computing the feature vector in the embedding space becomes problematic. The computation of the higher dimensional feature vector is proportional to d^{p}, whereas the computation of the kernel is proportional to $N_s d$, where N_s is the number of support vectors. So N_s would need to be larger than d^{p-1}, in order for the kernel approach to be more computationally intensive and this would normally only occur for $p \le 2$. The boundary in Fig. 2 was found with a non-homogeneous quadratic kernel.

A potential problem can occur when the data is not separable using a given kernel. An example of this is shown in Figs. 3 and 4, where the data cannot be separated with a linear kernel, due to an outlier. Then the assumptions leading to Eq. (1) no longer hold. To overcome this, one can introduce positive slack variables δ, which measure how far the points, which are on the wrong side of the boundary, are from the optimal separating hyperplanes. The constraint equation (2) then becomes,

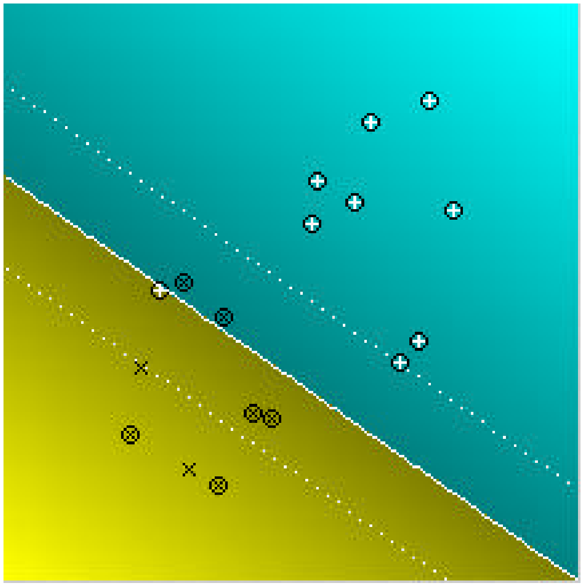

Fig. 3: Linearly non-separable data with C=∞.

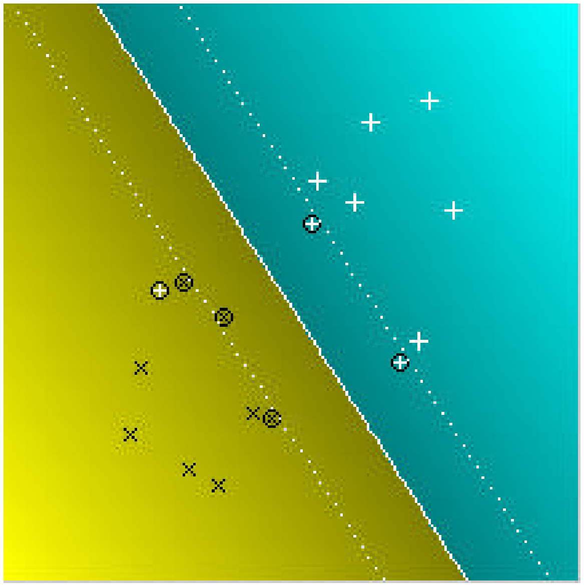

Fig. 4: Linearly non-separable data with C=10.

with a linear kernel. Notice the effect that the outlier had on the computed boundary and that nearly all the data points are support vectors. If we relax the error and choose *C*=10, then Fig. 4 results, whose separating hyperplane is closer to that of Fig. 1. In this case, the effect of the outlier has been minimized, and there are fewer support vectors required to define the boundary.

Digit	0	1	2	3						
0	96.9	0.1	1.1	0.1						
1	0.1	98.1	0.7	0.0						
2	0.1	0.4	91.1	1.2						
3	0.1	0.2	0.6	93.5						
4	0.4	0.1	0.8	0.1						
5	0.2	0.1	0.2	1.7						
6	0.8	0.3	1.5	0.1						
7	0.0	0.1	1.6	1.6	0.2	0.4	0.1	91.8	0.9	1.6
8	0.6	0.8	2.2	1.1	0.4	1.8	0.8	0.0	91.0	0.9
9	0.7	0.0	0.3	0.7	3.0	1.0	0.0	3.1	1.1	92.1

Table 1: Classification table for MNIST problem.

Vehicle	1	2	3	4	5
1	99.0	0.2	2.7	0.0	0.2
2	0.3	96.9	1.7	0.1	0.4
3	0.7	0.0	92.0	0.1	1.0
4	0.0	1.8	0.0	98.5	4.6
5	0.0	1.1	3.6	1.3	93.8

Table 2: Classification table for vehicles.

Results

In the past, we have used a variety of means to extract information from images, including statistical measures from multiresolution levels, with only limited success. We had considered using the entire image as a feature vector, instead of taking bits and pieces, but the dimensionality problem was an obstacle. The feature vector for a 32x32 image would be 1,024 elements long and would require a massive amount of training data to train a neural network, for example. However, the Support Vector Machine (SVM) has many nice properties that minimize the 'curse' of dimensionality. After extensively modifying an SVM toolbox[13], developed in MATLAB, we tested our algorithms on two data sets.

The first example was taken from the standard MNIST database[7] of handwritten digits, 0-9, which consists of a 60,000 image training set and a 10,000 image testing set. The database mixes two separate databases from NIST, one using Census Bureau employees and the other using high school students. There are approximately 250 different writers in the training set, who are different from the writers in the testing set. Samples of the images are shown in Fig. 5. As stated previously, we simply input the entire image into SVM as a feature vector. Because SVM is only a binary classifier, it is necessary to train ten separate classifiers for each digit versus all the remaining digits. To classify an image, one runs the sample through each classifier and chooses the one with the highest score. There are other ways to determine the classification from the output of each classifier[6], but we chose this 'winner take all' method for simplicity.

vectors are superfluous to the training. Training with this reduced set, gave us the parameters for the separating hyperplanes for each digit. We then used the entire 10,000 sample testing set to evaluate the classification properties of the SVM. The results of this ten class problem are shown in Table 1, where the correct values for the digits, 0-9, are in the first row and the SVM predictions are in the first column. The overall correct classification rate was over 93% on the testing set. This is quite good considering that the feature vector had 784 elements and we only used 800 training samples. In addition, note that the writing subjects in the testing set are different than in the training set. Other implementations of SVM obtain close to 1% error rates[7], when training with the full 60,000 sample training set.

Emboldened by our success with the digit classification, we turned to vehicle classification. We obtained (512x640) RGB digital images of five different vehicles (1 HMMWV and 4 trucks), taken from different positions about the front of the vehicles. To facilitate classification, we converted the images to grayscale and reduced the resolution to (16x20). Examples of the imagery, before and after, are shown in Fig. 6 and have been resized for display purposes. Since we had a limited number of images for each vehicle, we resorted to a jackknife approach for training, where we randomly chose 30 out of 50 images to train with and then tested with the remainder. We ran through 50 iterations of this and averaged the results, which are shown in Table 2. Again, the columns label the correct vehicle and the rows label the predicted vehicle. The overall classification rate was 96%.

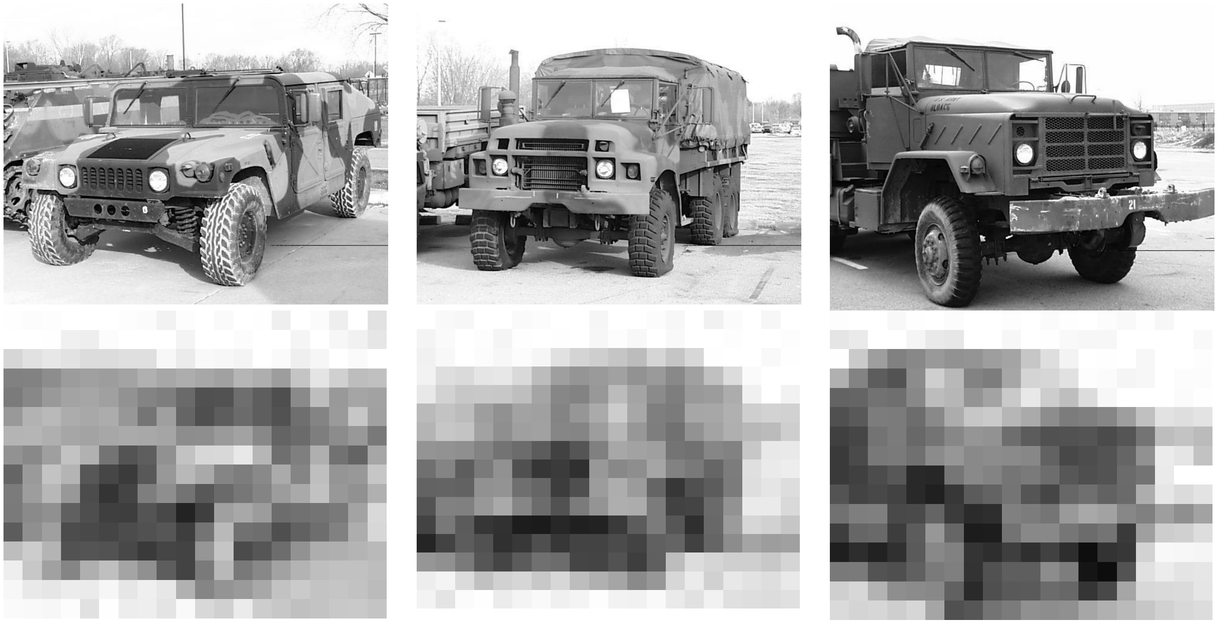

Fig. 6: Samples of vehicle imagery, original and reduced resolution.

(1998).

4) V. Vapnik, *The Nature of Statistical Learning*, Springer Verlag (1995).

5) C.J.C. Burges, ‘A tutorial on support vector machines for pattern recognition’, *Data Mining and Knowledge Discovery* **2**, 1 (1998).

6) B. Schölkopf, C.J.C. Burges, and A.J. Smola, eds., *Advances in Kernel Methods, Support Vector Learning,* MIT Press (1999).

7) Y. Lecun et al., ‘Comparison of learning algorithms for handwritten digit recognition’, *Proc. Int. Conf. Artificial Neural Networks* **II**, 53 (1995).

8) D.G. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley (1984).

9) R. Fletcher, *Practical Methods of Optimization*, John Wiley and Sons (1987).

10) D.P. Bertsekas, *Nonlinear Programming*, Athena Scientific (1995).

11) W. Karush, ‘Minima of functions of several variable with inequalities as side constraints’, Master’s Thesis, Dept. of Mathematics, Univ. of Chicago (1939).

12) H.W. Kuhn and A.W. Tucker, ‘Nonlinear programming’, *Proc. 2nd Berkeley Symp. On Mathematical Statistics and Probabalistics,* 481 (1951).

13) S. Gunn, ‘Support vector machines for classification and regression’, Technical Report, Image Speech and Intelligent Systems Group, Univ. of Southampton (1998).